## AMENDMENTS TO THE CLAIMS:

This listing of claims will replace all prior versions, and listings, of claims in the application:

### Listing of Claims:

1. (Currently Amended) A compiling system embodied on a computer readable storage medium for compiling a markup language file into an executable application, the compiling system comprising:

a parser for parsing the markup language file and providing the compiling system with detailed token information, wherein the markup language file is associated with at least one C# file;

a code generator for generating a language-independent tree of code expressions based on the token information, wherein the code generator receives the code in the at least one C# file inside the markup language file, and wherein the code expressions represent the markup file as a class; and

a compiler for compiling the code expressions to create the executable application, wherein the compiler determines an appropriate code provider for generating code in an appropriate language in response to the language-independent tree of code expressions.

2. (Original) The compiling system of claim 1, wherein the detailed token information comprises a tag.

3. (Original) The compiling system of claim 1, wherein the detailed token information comprises a property or event.

4. (Original) The compiling system of claim 1, wherein the detailed token information comprises a user code snippet.

5. (Canceled)

6. (Original) The compiling system of claim 5, wherein the compiler is configured to compile the markup language file and the code-behind file.

7. (Original) The compiling system of claim 1, wherein the executable application is an intermediate language application.

8. (Original) The compiling system of claim 1, further comprising a binary file generator for generating a binary file from non-code token information, wherein the binary file contains one record for each non-code token.

9. (Currently Amended) A compiling system embodied on a computer ~~storage~~ readable storage medium for compiling a markup language file into an executable application, the compiling system comprising:

a parser for parsing the markup language file and providing the compiling system with detailed token information including non-code token information to the compiling system, wherein the markup language file is associated with at least one C# ~~code-behind~~ file;

a binary file generator for generating a binary file from non-code token information, wherein the binary file contains one record for each non-code token;

a code generator for generating a language-independent code expression that represents the markup language file as a class; and

an application generator for compiling the code files into an executable application, wherein the application generator determines an appropriate code provider for generating code in an appropriate language in response to the language-independent tree of code expressions.

10. (Canceled)

11. (Previously Presented) The compiling system of claim 9, wherein the application generator combines the binary files into a single resource.

12. (Original) The compiling system of claim 9, wherein the detailed token information comprises a tag.

13. (Original) The compiling system of claim 9, wherein the detailed token information comprises a property or event.

14. (Original) The compiling system of claim 9, wherein the detailed token information comprises a user code snippet.

15. (Canceled)

16. (Previously Presented) The compiling system of claim 9, wherein the compiling system is configured to compile the markup language file and the code-behind file.

17. (Currently Amended) A method for compiling a markup language file into an executable application, the method comprising:

receiving a markup language file;

receiving a C# file, the C# file being associated with the received markup

language file;

parsing the markup language file into tokens and providing a compiling system with detailed information about the parsed tokens, the detailed information including inline code from the associated C# file ~~information~~;

receiving a command to create an intermediate language application;

upon receipt of the command to create an intermediate language application, generating a language-independent tree of code expressions based on the token information, wherein the code expressions represent the markup language file as a class; and

compiling the code expressions generated from the markup language file to create the executable application.

18. (Canceled)

19. (Canceled)

20. (Original) The method of claim 17, further comprising providing a tag as detailed token information.

21. (Original) The method of claim 17, further comprising providing a property or event as the detailed token information.

22. (Original) The method of claim 17, further comprising providing a user code snippet as the detailed token information.

23. (Canceled)

24. (Original) The method of claim 17, further comprising generating a binary file from non-code token information, wherein the binary file contains one record for each non-code token.

25. (Previously Presented) A computer readable storage medium storing the computer executable instructions for performing the method of claim 17.

26. (Currently Amended) A method for compiling a markup language file into an executable application, the method comprising:

receiving the markup language file;

receiving at least one code-behind file, wherein the at least one code-behind file contains a user code snippet and is associated with the received markup language file;

parsing the markup language file into tokens and providing a[[ the]] compiling system with detailed token information, wherein the detailed token information includes ~~including~~ non-code token information;

receiving a command to create an application or library containing a binary tokenized representation of the markup language file;

in response to receiving the command to create an application or library containing a binary tokenized representation of the markup language file, generating a binary file from the non-code token information, wherein the binary file contains one record for each non-code token;

generating a language-independent code expression that represents the markup language file as a class, wherein the generated language-independent

code expressions includes the user code snippet from the at least one code-behind file; and

compiling the code expressions into an executable application, wherein the compiling includes both the markup language file and the at least one code-behind file.

27. (Canceled)

28. (Previously Presented) The method of claim 26, further comprising combining the binary files into a single resource.

29. (Previously Presented) The method of claim 26, further comprising providing a tag as the detailed token information.

30. (Previously Presented) The method of claim 26, further comprising providing a property or event as the detailed token information.

31. (Canceled)

32. (Canceled)

33. (Previously Presented) A computer readable storage medium having computer executable instructions for performing the method of claim 26.